Discrete Structures for Computer Science

George Voutsadakis[1]

[1]Mathematics and Computer Science
Lake Superior State University

LSSU CSci 341

1. Regular Languages and Finite Automata
   - Regular Languages
   - Finite Automata
   - Regular Languages and Finite Automata
   - Regular Grammars
   - Properties of Regular Languages

Subsection 1

Regular Languages

Regular Languages

- Let A be a finite alphabet.
- Recall that a **language** L over A is a subset of A^*, i.e., a language is a set of strings.
- Recall also the following operations on languages:

$$\begin{array}{rcl} L \cup M & = & \{w \in A^* : w \in L \text{ or } w \in M\}; \\ LM & = & \{uv \in A^* : u \in L \text{ and } v \in M\} \\ L^* & = & \{\Lambda\} \cup L \cup L^2 \cup L^3 \cup \cdots . \end{array}$$

- The collection of **regular languages** over A is defined inductively as follows:

Basis: $\emptyset, \{\Lambda\}$ and $\{a\}$, $a \in A$, are regular languages;

Induction: If L and M are regular languages, then the following languages are also regular:

$$L \cup M, \quad LM, \quad L^*.$$

Example

- Let $A = \{a, b\}$ be an alphabet.
  Show that the following languages are regular:
  (a) $\{\Lambda, b\}$;
  (b) $\{a, ab\}$;
  (c) $\{\Lambda, b, bb, \ldots, b^n, \ldots\}$;
  (d) $\{a, ab, abb, \ldots, ab^n, \ldots\}$;
  (e) $\{\Lambda, a, b, aa, bb, \ldots, a^n, b^n, \ldots\}$.

  (a) By the basis $\{\Lambda\}, \{b\}$ are regular.
  By the induction $\{\Lambda, b\} = \{\Lambda\} \cup \{b\}$ is regular.
  (b) By the basis and (a), $\{a\}$ and $\{\Lambda, b\}$ are regular.
  By the induction $\{a\}\{\Lambda, b\} = \{a, ab\}$ is regular.
  (c) By the basis $\{b\}$ is regular.
  By the induction $\{b\}^* = \{\Lambda, b, b^2, \ldots\}$ is regular.
  (d) By the basis and (c), $\{a\}$ and $\{\Lambda, b, b^2, \ldots\}$ are regular.
  By the induction $\{a\}\{\Lambda, b, b^2, \ldots\} = \{a, ab, ab^2, \ldots\}$ is regular.
  (e) By (c), $\{\Lambda, a, a^2, \ldots\}$ and $\{\Lambda, b, b^2, \ldots\}$ are regular.
  So $\{\Lambda, a, a^2, \ldots\} \cup \{\Lambda, b, b^2, \ldots\} = \{\Lambda, a, b, a^2, b^2, \ldots\}$ is regular.

Regular Expressions

- The set of **regular expressions** over an alphabet A is defined inductively as follows:

Basis: $\Lambda, \emptyset$ and a, $a \in A$, are regular expressions;

Induction: If R and S are regular expressions, then the following expressions are also regular:

$$(R), \quad R + S, \quad R \cdot S, \quad R^*.$$

- Example: A sample of the infinitely many regular expressions over the alphabet $A = \{a, b\}$ are:

$$\begin{array}{ccccc} \Lambda, & \emptyset, & a, & b, & \Lambda + b, \\ b^*, & a + (b \cdot a), & (a + b) \cdot a, & a \cdot b^*, & a^* + b^*. \end{array}$$

Regular Expressions: Notational Conventions

- To avoid using too many parentheses, we assume that the operations are assigned priorities (from first to last):

$$* \qquad \cdot \qquad +$$

- Example: The regular expression $a + b \cdot a^*$ can be written in fully parenthesized form as

$$(a + (b \cdot (a^*))).$$

- We often use juxtaposition instead of $\cdot$ whenever no confusion arises.
- Example: We can write the preceding expression as

$$a + ba^*.$$

The Language of a Regular Expression

- To each regular expression E we associate a regular language $L(E)$ as follows, where A is an alphabet and R and S are regular expressions:

$$\begin{array}{rcl} L(\emptyset) & = & \emptyset; \\ L(\Lambda) & = & \{\Lambda\}; \\ L(a) & = & \{a\}, \quad a \in A; \\ L(R+S) & = & L(R) \cup L(S); \\ L(R \cdot S) & = & L(R)L(S); \\ L(R^*) & = & L(R)^*. \end{array}$$

- It is clear that through this association:
  - Each regular expression represents a regular language;
  - Each regular language is represented by a regular expression.

Example

- Find the language of the regular expression $a + bc^*$.

  We can evaluate the expression $L(a + bc^*)$ as follows:

$$
\begin{aligned}
L(a + bc^*) &= L(a) \cup L(bc^*) \\
&= L(a) \cup (L(b)L(c^*)) \\
&= L(a) \cup (L(b)L(c)^*) \\
&= \{a\} \cup (\{b\}\{c\}^*) \\
&= \{a\} \cup (\{b\}\{\Lambda, c, c^2, \ldots, c^n, \ldots\}) \\
&= \{a\} \cup \{b, bc, bc^2, \ldots, bc^n, \ldots\} \\
&= \{a, b, bc, bc^2, \ldots, bc^n, \ldots\}.
\end{aligned}
$$

Example

- Find regular expressions for the following languages:
  (a) $\{a, aa, aaa, \ldots, a^n, \ldots\}$;
  (b) $\{\Lambda, a, b, ab, abb, abbb, \ldots, ab^n, \ldots\}$.

(a) We have

$$\begin{array}{rcl}\{a, aa, aaa, \ldots, a^n, \ldots\} & = & \{a\}\{\Lambda, a, a^2, \ldots\} \\ & = & \{a\}\{a\}^* = L(a) \cdot L(a)^* \\ & = & L(a) \cdot L(a^*) = L(a \cdot a^*).\end{array}$$

So the regular expression is $a \cdot a^*$.

(b) We have

$$\begin{array}{l}\{\Lambda, a, b, ab, abb, abbb, \ldots, ab^n, \ldots\} \\ = \{\Lambda\} \cup \{b\} \cup \{a, ab, abb, abbb, \ldots\} \\ = \{\Lambda\} \cup \{b\} \cup \{a\}\{\Lambda, b, bb, bbb, \ldots\} \\ = \{\Lambda\} \cup \{b\} \cup \{a\}\{b\}^* = L(\Lambda) \cup L(b) \cup L(a)L(b)^* \\ = L(\Lambda) \cup L(b) \cup L(ab^*) = L(\Lambda + b + ab^*).\end{array}$$

So the regular expression is $\Lambda + b + ab^*$.

Equality of Regular Expressions

- We say that two regular expressions R and S are **equal**, written $R = S$, if they represent the same languages, i.e.,

$$R = S \text{ if and only if } L(R) = L(S).$$

- Example: We have $a + b = b + a$.

  This follows from the equality

$$\begin{aligned} L(a+b) &= L(a) \cup L(b) = \{a\} \cup \{b\} = \{a, b\} \\ &= \{b\} \cup \{a\} = L(b) \cup L(a) = L(b+a). \end{aligned}$$

- Example: We have $ab \neq ba$.

  This follows from

$$\begin{aligned} L(ab) &= L(a)L(b) = \{a\}\{b\} = \{ab\}; \\ L(ba) &= L(b)L(a) = \{b\}\{a\} = \{ba\}. \end{aligned}$$

Properties of Regular Expressions

- **Properties of Regular Expressions**

  (+) $R + T = T + R$;
  $R + \emptyset = \emptyset + R = R$;
  $R + R = R$;
  $(R + S) + T = R + (S + T)$.

  $(\cdot)$ $R\emptyset = \emptyset R = \emptyset$;
  $R\Lambda = \Lambda R = R$;
  $(RS)T = R(ST)$.

  (Dist) $R(S + T) = RS + RT$;
  $(S + T)R = SR + TR$.

Properties of Regular Expressions (Cont'd)

- **Properties Involving Closure**

$$\emptyset^* = \Lambda^* = \Lambda;$$
$$R^* = R^*R^* = (R^*)^* = R + R^*;$$
$$R^* = \Lambda + R^* = (\Lambda + R)^* = (\Lambda + R)R^* = \Lambda + RR^*;$$
$$R^* = (R + \cdots + R^k)^*,\ k \geq 1;$$
$$R^* = \Lambda + R + \cdots + R^{k-1} + R^kR^*,\ k \geq 1;$$
$$R^*R = RR^*;$$
$$(R + S)^* = (R^* + S^*)^* = (R^*S^*)^* = (R^*S)^*R^* = R^*(SR^*)^*;$$
$$R(SR)^* = (RS)^*R;$$
$$(R^*S)^* = \Lambda + (R + S)^*S;$$
$$(RS^*)^* = \Lambda + R(R + S)^*.$$

Selected Proofs

- Show the following:
  (a) $R + R = R$;
  (b) $R(S + T) = RS + RT$;
  (c) $R^* = R^* R^*$.

(a) $L(R + R) = L(R) \cup L(R) = L(R)$;

(b) We have $L(R(S + T))$

$$\begin{aligned} &= L(R)L(S + T) = L(R)(L(S) \cup L(T)) \\ &= L(R)L(S) \cup L(R)L(T) = L(RS) \cup L(RT) = L(RS + RT). \end{aligned}$$

(c) We show $L(R^*) \subseteq L(R^* R^*)$ and $L(R^* R^*) \subseteq L(R^*)$.

- Suppose $x \in L(R^*) = L(R)^*$. Then $x = x\Lambda \in L(R)^* L(R)^* = L(R^* R^*)$. Thus $L(R^*) \subseteq L(R^* R^*)$.
- Suppose, conversely, that $x \in L(R^* R^*) = L(R)^* L(R)^*$. Thus, $x = yz$, with $y \in L(R)^*$ and $z \in L(R)^*$. But then $x = yz$, with $y \in L(R)^m$ and $z \in L(R)^n$, for some $m, n \in \mathbb{N}$. So $x = yz \in L(R)^{m+n} \subseteq L(R)^*$. This shows that $L(R^* R^*) \subseteq L(R^*)$.

Proving Equality of Regular Expressions

- Prove the following equality:

$$ba^*(baa^*)^* = b(a + ba)^*.$$

Since both expressions start with the letter b, it suffices to show the simpler equality obtained by canceling b from both sides:

$$a^*(baa^*)^* = (a + ba)^*.$$

By the properties, we know that $(R + S)^* = R^*(SR^*)^*$, for any regular expressions R and S.

In particular, for $R = a$ and $S = ba$, we get

$$(a + ba)^* = a^*(baa^*)^*.$$

Therefore the given equation is true.

Example

- Show $(\emptyset + a + b)^* = a^*(ba^*)^*$.

  We start with the left side as follows:

$$\begin{array}{rcll} (\emptyset + a + b)^* & = & (a+b)^* & (\emptyset + R = R) \\ & = & a^*(ba^*)^* & ((R+S)^* = R^*(SR^*)^*) \end{array}$$

Example

- Show that $b^*(abb^* + aabb^* + aaabb^*)^* = (b + ab + aab + aaab)^*$.
  We have

$$\begin{aligned} & b^*(abb^* + aabb^* + aaabb^*)^* \\ & = b^*((ab + aab + aaab)b^*)^* \quad ((S + T)R = SR + TR) \\ & = (b + ab + aab + aaab)^* \quad (R^*(SR^*)^* = (R + S)^*) \end{aligned}$$

Example

- Show $R + RS^*S = a^*bS^*$, where $R = b + aa^*b$ and S is any regular expression.

  We have:

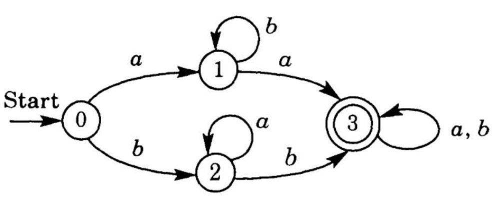
$$\begin{array}{lcll} R + RS^*S & = & R\Lambda + RS^*S & (R = R\Lambda) \\ & = & R(\Lambda + S^*S) & (R(S+T) = RS + RT) \\ & = & R(\Lambda + SS^*) & (R^*R = RR^*) \\ & = & RS^* & (R^* = \Lambda + RR^*) \\ & = & (b + aa^*b)S^* & (R = b + aa^*b) \\ & = & (\Lambda b + aa^*b)S^* & (\Lambda R = R) \\ & = & (\Lambda + aa^*)bS^* & ((S+T)R = SR + TR) \\ & = & a^*bS^* & (R^* = \Lambda + RR^*) \end{array}$$

Subsection 2

Finite Automata

Deterministic Finite Automata

- A **deterministic finite automaton** (**DFA**) is quintuple $M = \langle A, S, s_0, F, \delta \rangle$, consisting of:
  - A set A,called the **input alphabet**;
  - A set S, called the set of **states**;
  - $s_0 \in S$, called the **start** or **initial state**;
  - $F \subseteq S$, called the **set of final states**;
  - A function $\delta : S \times A \to S$, called the (**state**) **transition function**.
- Example: Let $M = \langle A, S, s_0, F, \delta \rangle$, where
  - $A = \{a, b\}$;
  - $S = \{0, 1, 2, 3\}$;
  - $s_0 = 0$;
  - $F = \{3\}$;



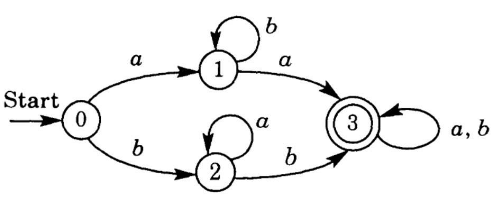
  - $\delta(0, a) = 1$, $\delta(0, b) = 2$, $\delta(1, a) = 3$, $\delta(1, b) = 1$, $\delta(2, a) = 2$, $\delta(2, b) = 3$, $\delta(3, a) = 3$, $\delta(3, b) = 3$.

Language Accepted by a DFA

- Let M be a DFA with input alphabet A.
- The DFA M **accepts** a string w in A^* if there is a path from the start state to some final state such that w is the concatenation of the labels on the edges of the path.
- Otherwise, the DFA **rejects** w.
- The set of all strings accepted by a DFA M is called the **language of** M and is denoted by $L(M)$.

Example

- Consider again the DFA M shown in the figure:

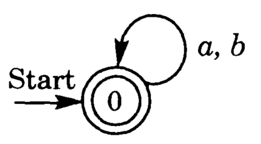


- This DFA:
  - accepts the string aba;
  - accepts the string $baaabab$;
  - accepts infinitely many strings because we can traverse the loop out of and into states $1, 2$ or 3 any numbers of times;
  - rejects infinitely many strings, e.g., any string of the form ab^n.

Example

- Give a DFA that recognizes the language $(a+b)^*$.

  Provide both pictorial and formal descriptions.
- The DFA recognizing $(a+b)^* = \{a,b\}^*$ (all strings over $\{a,b\}$) is



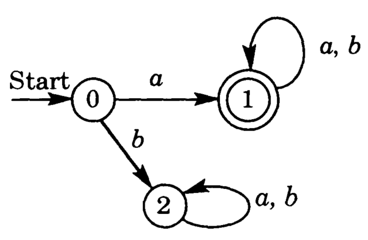
Its formal description is $M = \langle A, S, s_0, F, \delta \rangle$, with
  - $A = \{a,b\}$;
  - $S = \{0\}$;
  - $s_0 = 0$;
  - $F = \{0\}$;
  - $\delta(0,a) = \delta(0,b) = 0$.

Example

- Give a DFA that recognizes the language $a(a+b)^*$.

  Provide both pictorial and formal descriptions.

  The DFA recognizing $a(a+b)^* = \{a\} \cdot \{a,b\}^*$ (all strings over $\{a,b\}$ starting with a) is



state s	letter ℓ	$\delta(s,\ell)$
0	a	1
0	b	2
1	a	1
1	b	1
2	a	2
2	b	2

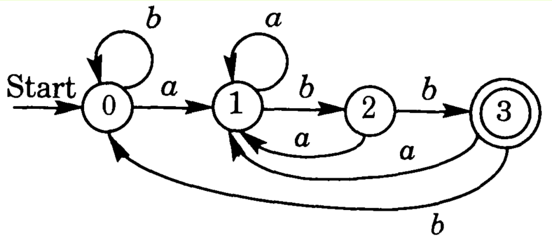
It is $M = \langle A, S, s_0, F, \delta \rangle$, with $A = \{a,b\}$, $S = \{0,1,2\}$, $s_0 = 0$, $F = \{1\}$, and $\delta(0,a) = 1$, $\delta(0,b) = 2$, $\delta(1,a) = \delta(1,b) = 1$, $\delta(2,a) = \delta(2,b) = 2$.

Example

- Build a DFA to recognize the regular language represented by the regular expression $(a+b)^*abb$ over the alphabet $A = \{a, b\}$.

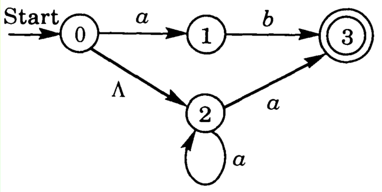
  The language is the set of strings that begin with anything but must end with the string *abb*.

  A DFA that recognizes it is



Nondeterministic Finite Automata

- A **nondeterministic finite automaton** (**NFA**) is quintuple $N = \langle A, S, s_0, F, \delta \rangle$, consisting of:
  - A set A,called the **input alphabet**;
  - A set S, called the set of **states**;
  - $s_0 \in S$, called the **start** or **initial state**;
  - $F \subseteq S$, called the **set of final states**;
  - A function $\delta : S \times A \cup \{\Lambda\} \to \mathcal{P}(S)$, called the **transition function**.
- Example: Let $M = \langle A, S, s_0, F, \delta \rangle$, where
  - $A = \{a, b\}$;
  - $S = \{0, 1, 2, 3\}$;
  - $s_0 = 0$;
  - $F = \{3\}$;



  - $\delta(0, a) = \{1\}$, $\delta(0, b) = \emptyset$, $\delta(0, \Lambda) = \{2\}$, $\delta(1, a) = \emptyset$, $\delta(1, b) = \{3\}$, $\delta(1, \Lambda) = \emptyset$, $\delta(2, a) = \{2, 3\}$, $\delta(2, b) = \emptyset$, $\delta(2, \Lambda) = \emptyset$, $\delta(3, a) = \emptyset$, $\delta(3, b) = \emptyset$, $\delta(3, \Lambda) = \emptyset$.

Language Accepted by an NFA

- Let A be an alphabet and N be an NFA with input alphabet A.
- The NFA N **accepts** a string w in A^* if there exists a path from the start state to some final state such that w is the concatenation of the labels on the edges of the path.
- Otherwise, the NFA **rejects** w.
- The **language** of the NFA N is the set of strings that it accepts, denoted by $L(N)$.

NFAs versus DFAs

- Note that the only difference in the definitions of an NFA versus that of a DFA lies in the transition function:
  - For a DFA $\delta : S \times A \to S$;
  - For an NFA $\delta' : S \times A \cup \{\Lambda\} \to \mathcal{P}(S)$.
- Therefore, any DFA with transition function $\delta : S \times A \to S$ can be viewed as an NFA by defining $\delta' : S \times A \cup \{\Lambda\} \to \mathcal{P}(S)$ by:

$$\begin{array}{rcl} \delta'(s,a) & = & \{\delta(s,a)\}, \text{ for all } s \in S, a \in A; \\ \delta'(s,\Lambda) & = & \emptyset, \text{ for all } s \in S. \end{array}$$

Example

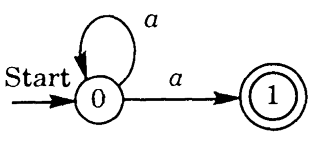
- Build an NFA that recognizes the language a^*a over the alphabet $A = \{a\}$.

  Give both a pictorial and a formal description.

  The language is $L(a^*a) = \{a\}^* \cdot \{a\} = \{a, aa, aaa, \ldots\}$.

  The key is to ensure that a string is accepted if and only if it contains at least one a.

  An NFA that does the job is

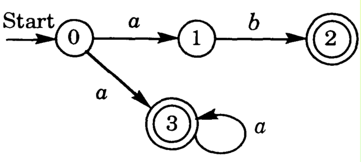


state s	letter ℓ	$\delta(s, \ell)$
0	a	$\{0, 1\}$
0	Λ	$\emptyset$
1	a	$\emptyset$
1	Λ	$\emptyset$

  We let $N = \langle A, S, s_0, F, \delta \rangle$, where $A = \{a\}$, $S = \{0, 1\}$, $s_0 = 0$, $F = \{1\}$ and $\delta(0, a) = \{0, 1\}$, $\delta(\Lambda) = \emptyset$, $\delta(1, a) = \delta(1, \Lambda) = \emptyset$.

Example

- Build an NFA that recognizes the language $ab + a^*a$ over the alphabet $A = \{a, b\}$.
  Give both a pictorial and a formal description.
  The language is
  $L(ab + a^*a) = \{ab\} \cup (\{a\}^* \cdot \{a\}) = \{ab, a, aa, aaa, \ldots\}$.
  An NFA that does the job is

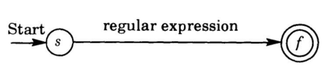


We let $N = \langle A, S, s_0, F, \delta \rangle$, where $A = \{a, b\}$, $S = \{0, 1, 2, 3\}$, $s_0 = 0$, $F = \{2, 3\}$ and $\delta(0, a) = \{1, 3\}$, $\delta(0, b) = \emptyset$, $\delta(0, \Lambda) = \emptyset$, $\delta(1, a) = \emptyset$, $\delta(1, b) = \{2\}$, $\delta(1, \Lambda) = \emptyset$, $\delta(2, a) = \emptyset$, $\delta(2, b) = \emptyset$, $\delta(2, \Lambda) = \emptyset$, $\delta(3, a) = \{3\}$, $\delta(3, b) = \emptyset$, $\delta(3, \Lambda) = \emptyset$.

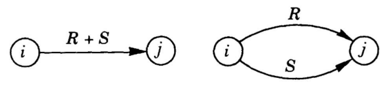
Subsection 3

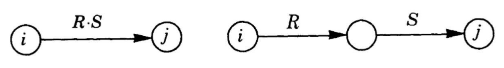
Regular Languages and Finite Automata

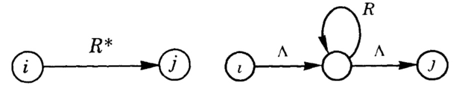
From a Regular Expression to an NFA (Top-Down)

- Given a regular expression over the alphabet A.
- Construct the following machine: 

- Transform this machine into an NFA by applying the following rules until all edges are labeled with either a letter in A or Λ:
  - If an edge is labeled by $\emptyset$, erase the edge;
  - Transform any diagram as on the left to one as on the right:

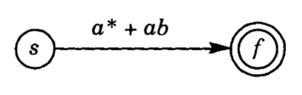


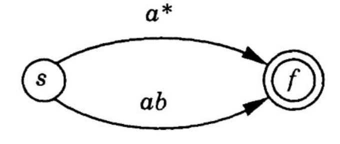


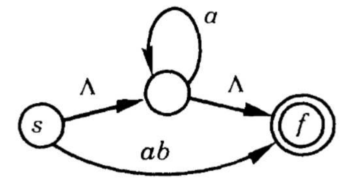


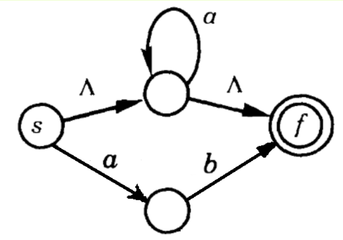
Example

- Construct an NFA for the regular expression $a^* + ab$.



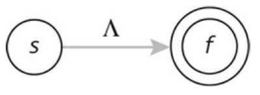


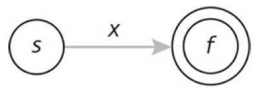




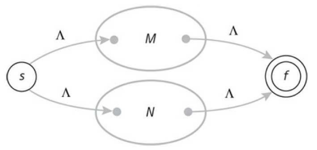
From a Regular Expression to an NFA (Bottom-Up)

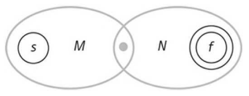
- Apply the following rules inductively to any regular expression, where the letters s and f represent the start state and the final state:
  1. Construct an NFA of the following form:
     - left, for each occurrence of the symbol $\emptyset$ in the regular expression;
     - center, for each occurrence of the symbol Λ in the regular expression;
     - right, for each occurrence of a letter x in the regular expression.

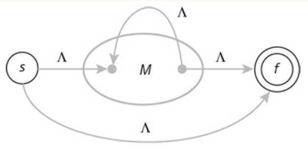




  2. Let M and N be NFAs for the regular expressions R and S, respectively. We construct NFAs for the regular expressions $R + S$, RS and R^*:

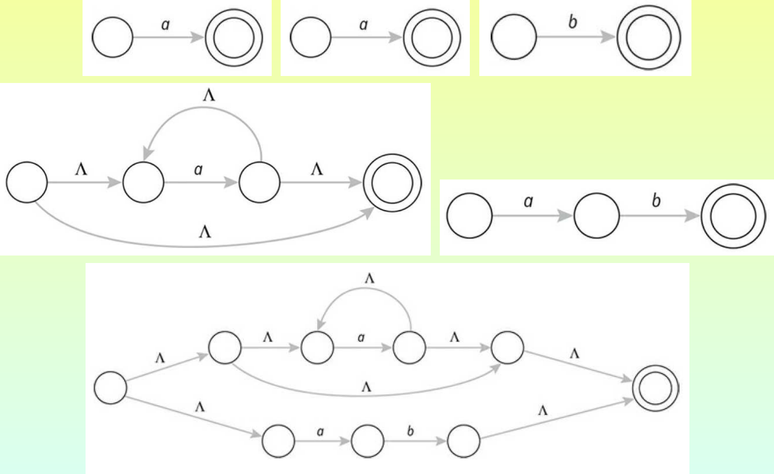






Example

- Use the bottom-up algorithm to construct an NFA for the regular expression $a^* + ab$.



Transforming an NFA to a Regular Expression

- Given a DFA or an NFA.
  1. Create a new start state s, and draw a new edge labeled with Λ from s to the original start state.
  2. Create a new final state f, and draw new edges labeled with Λ from all the original final states to f.
  3. For each pair of states i and j that have more than one edge from i to j, replace all the edges from i to j by a single edge labeled with the regular expression formed by the sum of the labels on each of the edges from i to j.
  4. Construct a sequence of new machines by eliminating one state at a time until the only states remaining are s and f.
     As each state is eliminated, a new machine is constructed from the previous machine following the process described in the next slide.

NFA to a Regular Expression: Eliminating a State

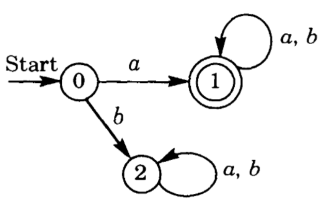
- **Eliminate State** k:
  - Let $\text{old}(i,j)$ denote the label on edge (i,j) of the current machine. If there is no edge (i,j), then set $\text{old}(i,j)=\emptyset$.
  - For each pair of edges (i,k) and (k,j), where $i\neq k$ and $j\neq k$, calculate a new edge label, $\text{new}(i,j)$, as follows:

$$\text{new}(i,j)=\text{old}(i,j)+\text{old}(i,k)\text{old}(k,k)^{*}\text{old}(k,j).$$

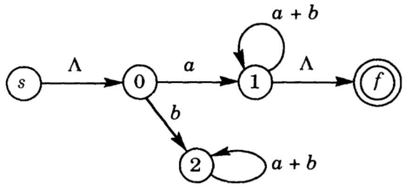
  - For all other edges (i,j), $i\neq k$ and $j\neq k$, set $\text{new}(i,j)=\text{old}(i,j)$.
  - The states of the new machine are those of the current machine with state k eliminated.
  - The edges of the new machine are those edges (i,j) for which label $\text{new}(i,j)$ has been calculated.
- At the end s and f are the two remaining states:
  - If there is an edge (s,f), then the regular expression $\text{new}(s,f)$ represents the language of the original automaton.
  - If there is no edge (s,f), then the language of the original automaton is empty, which is signified by the regular expression $\emptyset$.

Example

- Find a regular expression for the language accepted by the DFA given below:

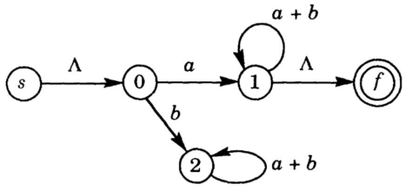


- In the first step we transform the DFA by adding a new start state and a new final state.



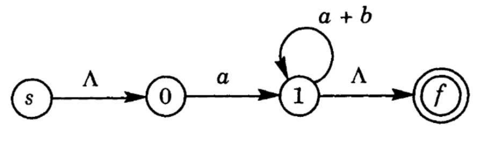
Example (Cont'd)

- We have the NFA



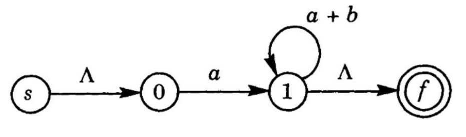
- We eliminate state 2.

  There are no paths passing through state 2 between states that are adjacent to state 2. So new$(i,j) =$ old(i,j) for each pair of states (i,j), where $i \neq 2$ and $j \neq 2$.



Example (Cont'd)

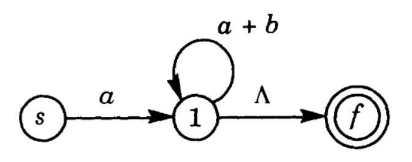
- We have the NFA



- We eliminate state 0.

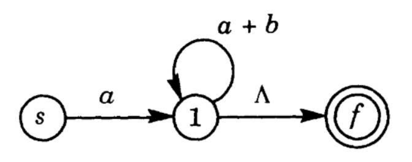
  We add a new edge $(s, 1)$, labeled with the regular expression

$$\begin{aligned} \text{new}(s,1) &= \text{old}(s,1) + \text{old}(s,0)\text{old}(0,0)^*\text{old}(0,1) \\ &= \emptyset + \Lambda\emptyset^* a = a. \end{aligned}$$



Example (Cont'd)

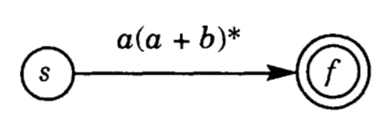
- We have the NFA



- We eliminate state 1.

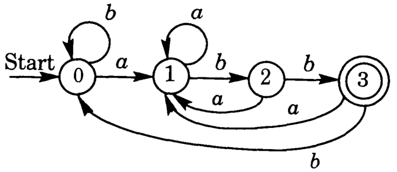
  We add a new edge $\langle s, f \rangle$, labeled with the regular expression

$$\begin{aligned} \text{new}(s,f) &= \text{old}(s,f) + \text{old}(s,1)\text{old}(1,1)^*\text{old}(1,f) \\ &= \emptyset + a(a+b)^*\Lambda = a(a+b)^*. \end{aligned}$$

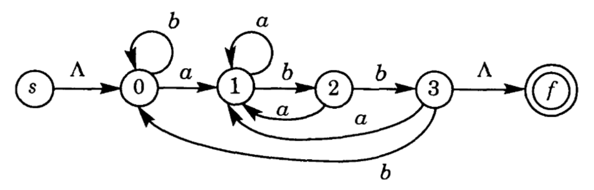


Example

- Find a regular expression for the language accepted by the DFA

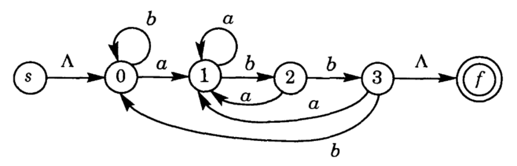


- In the first step we transform the DFA by adding a new start state s and a new final state f.



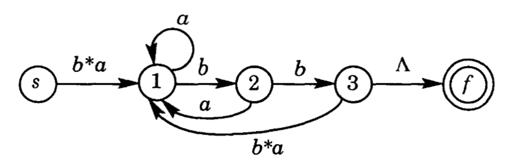
Example (Cont'd)

- We have the NFA



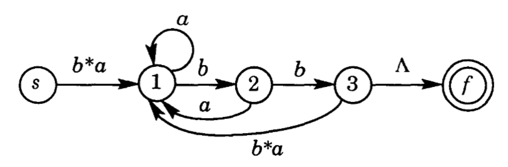
- We eliminate state 0.
  We have the following:

$$\begin{aligned} \text{new}(s,1) &= \emptyset + \Lambda b^* a = b^* a; \\ \text{new}(3,1) &= a + bb^* a = (\Lambda + bb^*)a = b^* a. \end{aligned}$$



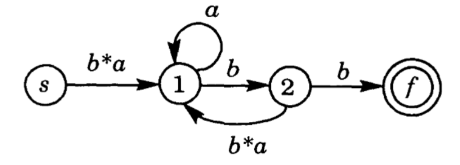
Example (Cont'd)

- We have the NFA



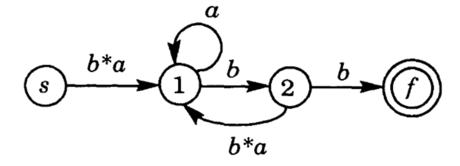
- We eliminate state 3.
  We have the following:

$$\begin{array}{rcl} \text{new}(2,f) & = & \emptyset + b\emptyset^*\Lambda = b; \\ \text{new}(2,1) & = & a + b\emptyset^* b^* a = a + bb^* a = (\Lambda + bb^*)a = b^* a. \end{array}$$



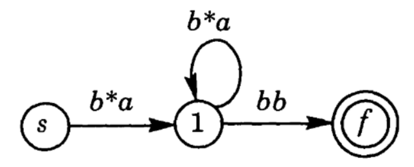
Example (Cont'd)

- We have the NFA



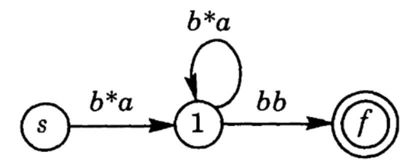
- We eliminate state 2.
  We have the following:

$$\begin{array}{rcl} \text{new}(1,f) & = & \emptyset + b\emptyset^* b = bb; \\ \text{new}(1,1) & = & a + b\emptyset^* b^* a = a + bb^* a = (\Lambda + bb^*)a = b^* a. \end{array}$$



Example (Cont'd)

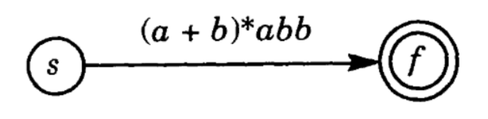
- We have the NFA



- We eliminate state 1.

  We have the following:

$$\begin{aligned} \text{new}(s,f) &= \emptyset + b^* a (b^* a)^* bb = b^* (ab^*)^* abb \quad (R(SR)^* = (RS)^* R) \\ &= b^* (ab^*)^* abb = (a+b)^* abb. \quad ((R+S)^* = R^* (SR^*)^*) \end{aligned}$$



Lambda Closure of a State in an NFA

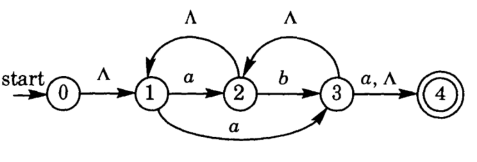
- Let N be an NFA and s be one of its states.
- The **lambda closure** of s, denoted $\lambda(s)$, is the set of states that can be reached from s by traversing zero or more λ edges.
- We define $\lambda(s)$ inductively as follows for any state s in N:

Basis: $s \in \lambda(s)$;

Induction: If $p \in \lambda(s)$ and there is a Λ edge from p to q, then $q \in \lambda(s)$.

Example

- Consider the following NFA $N = \langle A, S, s_0, F, \delta \rangle$, which is described both in graphical form and formally:



δ	a	b	Λ
0	$\emptyset$	$\emptyset$	$\{1\}$
1	$\{2,3\}$	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\{1\}$
3	$\{4\}$	$\emptyset$	$\{2,4\}$
4	$\emptyset$	$\emptyset$	$\emptyset$

The lambda closures for the five states of the NFA are as follows:

$$\begin{array}{lcl} \lambda(0) & = & \{0,1\}; \\ \lambda(1) & = & \{1\}; \\ \lambda(2) & = & \{1,2\}; \\ \lambda(3) & = & \{1,2,3,4\}; \\ \lambda(4) & = & \{4\}. \end{array}$$

Lambda Closure of a Set of States in an NFA

- Let N be an NFA and S be a set of states.
- The **lambda closure** of S, denoted $\lambda(S)$, is the set of states that can be reached from states in S by traversing zero or more Λ edges.
- If C and D are any sets of states, then we have

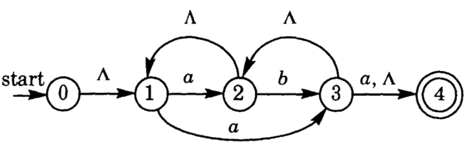
$$\lambda(C \cup D) = \lambda(C) \cup \lambda(D).$$

- More generally, the lambda closure of a union of sets is the union of the lambda closures of the sets.
- This property allows computing the lambda closure of a set by calculating the union of the lambda closures of the individual elements in the set:

$$\lambda(\{s_1, s_2, \ldots, s_n\}) = \lambda(s_1) \cup \lambda(s_2) \cup \cdots \cup \lambda(s_n).$$

Example

- Consider again the NFA $N = \langle A, S, s_0, F, \delta \rangle$, shown below:



We computed

$$\begin{array}{rcl} \lambda(0) & = & \{0,1\}; \\ \lambda(1) & = & \{1\}; \\ \lambda(2) & = & \{1,2\}; \\ \lambda(3) & = & \{1,2,3,4\}; \\ \lambda(4) & = & \{4\}. \end{array}$$

Therefore

$$\lambda(\{0,2,4\}) = \lambda(0) \cup \lambda(2) \cup \lambda(4) = \{0,1\} \cup \{1,2\} \cup \{4\} = \{0,1,2,4\}.$$

Transforming an NFA to a DFA

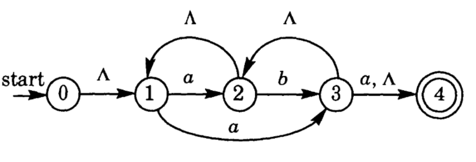
- Given an NFA over alphabet A with transition function δ.
- We construct a DFA over A with transition function δ' that accepts the same language as the NFA.
- The states of the DFA are represented as certain subsets of NFA states.
  1. The DFA start state is $\lambda(s)$, where s is the NFA start state. Perform Step 2 for this DFA start state.
  2. If $\{s_1, \ldots, s_n\}$ is a DFA state and $a \in A$, then construct the following DFA state and part of the transition function δ' in either of two ways:

$$\begin{aligned}\delta'(\{s_1, \ldots, s_n\}, a) &= \lambda(\delta(s_1, a) \cup \cdots \cup \delta(s_n, a)) \\ &= \lambda(\delta(s_1, a)) \cup \cdots \cup \lambda(\delta(s_n, a)).\end{aligned}$$

     Repeat Step 2 for each new DFA state constructed in this way.
  3. A DFA state is final if one of its elements is an NFA final state.

Example

- Use the preceding algorithm with input the NFA pictured below to obtain a DFA accepting the same language.



We construct step-by-step the transition table for the DFA:

	δ'	a	b
Start	$\{0,1\}$	$\{1,2,3,4\}$	$\emptyset$
Final	$\{1,2,3,4\}$	$\{1,2,3,4\}$	$\{1,2,3,4\}$
	$\emptyset$	$\emptyset$	$\emptyset$

$$\begin{aligned}
\delta'(\{0,1\},a) &= \lambda(\delta(0,a)\cup\delta(1,a)) = \lambda(\emptyset\cup\{2,3\}) \\
&= \lambda(\{2,3\}) = \lambda(2)\cup\lambda(3) \\
&= \{1,2\}\cup\{1,2,3,4\} = \{1,2,3,4\}.
\end{aligned}$$

Example (Cont'd)

- We came up with the DFA having transition table:

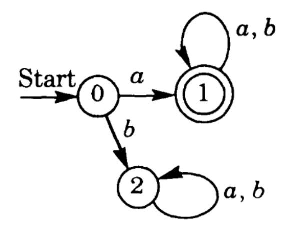
	δ'	a	b
Start	$\{0, 1\}$	$\{1, 2, 3, 4\}$	$\emptyset$
Final	$\{1, 2, 3, 4\}$	$\{1, 2, 3, 4\}$	$\{1, 2, 3, 4\}$
	$\emptyset$	$\emptyset$	$\emptyset$

- We rename states for elegance:

$$\text{Replace: } \{0, 1\} \text{ by } 0, \quad \{1, 2, 3, 4\} \text{ by } 1, \quad \emptyset \text{ by } 2.$$

- Then the table becomes as on the left yielding the DFA pictured.

	δ'	a	b
Start	0	1	2
Final	1	1	1
	2	2	2



Fundamental Theorems on Finite Automata

- Given a regular language, we can construct a regular expression whose language is the given language.

  Conversely, given a regular expression, its language is a regular language.
- DFAs are special cases of NFAs.

  Given an NFA, we may construct a DFA whose language is the same as that of the given NFA.
- Given an NFA we may construct a regular expression whose language is that accepted by the given NFA.

  Conversely, given a regular expression, we can construct an NFA that accepts the language of the original regular expression.
- In Summary:
  - Regular expressions represent regular languages;
  - (**Kleene**) DFAs recognize the regular languages;
  - (**Rabin** and **Scott**) NFAs recognize the regular languages.

Subsection 4

Regular Grammars

Regular Grammars

- A grammar is called a **regular grammar** if each production takes one of the following forms, where the capital letters are nonterminals and w is a string of terminals:

$$\begin{array}{lcl} S & \to & \Lambda; \\ S & \to & w; \\ S & \to & T; \\ S & \to & wT. \end{array}$$

- So only one nonterminal can appear on the right side of a production, and it must appear at the right end of the right side.
- Example:
  - The productions $A \to aBc$ and $S \to TU$ are not part of a regular grammar.
  - The production $A \to abcA$ is admissible.

Regular Expressions and Regular Grammars

- Each line of the following list describes a regular language in terms of a regular expression and a regular grammar:

Regular Expression	Regular Grammar
a^*	$S \to \Lambda \| aS$
$(a+b)^*$	$S \to \Lambda \| aS \| bS$
$a^* + b^*$	$S \to \Lambda \| A \| B$ $A \to a \| aA$ $B \to b \| bB$
$a^* b$	$S \to b \| aS$
ba^*	$S \to bA$ $A \to \Lambda \| aA$
$(ab)^*$	$S \to \Lambda \| abS$

Grammars for Products of Languages

- Most problems occur in trying to construct a regular grammar for a language that is the product of languages.
- Example: Construct a regular grammar for the language of the regular expression a^*bc^*.

  Observe that the strings of a^*bc^* start with either a or b.

  We can represent this property by writing down the following two productions, where S is the start symbol: $S \to aS|bC$.

  Now we need a definition for C to derive the language of c^*.

  The following two productions do the job:

  $$C \to \Lambda|cC.$$

  Therefore a regular grammar for a^*bc^* can be written as follows:

  $$\begin{array}{rcl} S & \to & aS|bC \\ C & \to & \Lambda|cC. \end{array}$$

Example

- We consider some regular languages, all of which consist of strings of a's followed by strings of b's.
- The largest language of this form is the language $\{a^m b^n : m, n \in \mathbb{N}\}$, which is represented by the regular expression $a^* b^*$.
- A regular grammar for this language can be written as follows:

$$\begin{array}{rcl} S & \to & \Lambda | aS | B \\ B & \to & b | bB \end{array}$$

Example (Cont'd)

- We look at four sublanguages of $\{a^m b^n : m, n \in \mathbb{N}\}$:

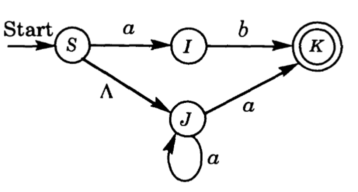
Language	Expression	Grammar
$\{a^m b^n : m \geq 0, n > 0\}$	$a^* bb^*$	$S \to aS \mid B$ $B \to b \mid bB$
$\{a^m b^n : m > 0, n \geq 0\}$	$aa^* b^*$	$S \to aA$ $A \to aA \mid B$ $B \to \Lambda \mid bB$
$\{a^m b^n : m > 0, n > 0\}$	$aa^* bb^*$	$S \to aA$ $A \to aA \mid B$ $B \to b \mid bB$
$\{a^m b^n : m > 0 \text{ or } n > 0\}$	$aa^* b^* + a^* bb^*$	$S \to aA \mid bB$ $A \to \Lambda \mid aA \mid B$ $B \to \Lambda \mid bB$

Transforming an NFA to a Regular Grammar

- Given an NFA.
- Perform the following steps to construct a regular grammar that generates the language of a given NFA:
  1. Rename the states to a set of capital letters;
  2. The start symbol is the NFA's start state;
  3. For each transition from I to J labeled with a, create the production $I \to aJ$;
  4. For each state transition from I to J labeled with Λ, create the production $I \to J$;
  5. For each final state K, create the production $K \to \Lambda$.

Example

- Consider the NFA shown below:



Construct a regular grammar whose language is the same as that accepted by the NFA.

The regular grammar has start symbol S.

It consists of the following productions:

$$\begin{array}{rcl} S & \to & aI \mid J \\ I & \to & bK \\ J & \to & aJ \mid aK \\ K & \to & \Lambda \end{array}$$

Regular Grammar: Standard Form

- Suppose G is a regular grammar.
- It is said to be in **standard form** if all its productions have one of two forms
$$S \to x \qquad S \to xT,$$
where x is either Λ or a single letter.
- Any regular grammar can be converted into one in standard form.
- Example: If we have a production like
$$A \to bcB$$
we replace it by the following two productions, where C is a new (not already occurring in the grammar) nonterminal:
$$A \to bC \quad \text{and} \quad C \to cB.$$

Transforming a Regular Grammar to an NFA

- Given a regular grammar.
- Perform the following steps to construct an NFA that accepts the language of the given regular grammar:
    1. If necessary, transform the grammar to standard form;
    2. The start state of the NFA is the grammar's start symbol;
    3. For each production $I \to aJ$, construct a state transition from I to J labeled with the letter a;
    4. For each production $I \to J$, construct a state transition from I to J labeled with Λ;
    5. If there are productions of the form $I \to a$ for some letter a, then create a single new state symbol F.
       For each production $I \to a$, construct a state transition from I to F labeled with a.
    6. The final states of the NFA are F together with all I for which there is a production $I \to \Lambda$.

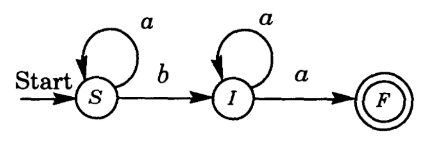
Example

- Transform the following regular grammar into an NFA that accepts the language of the grammar:

$$\begin{array}{rcl} S & \rightarrow & aS|bI \\ I & \rightarrow & a|aI \end{array}$$

Since there is a production $I \rightarrow a$, we introduce a new state F.

The NFA is shown below:



Subsection 5

Properties of Regular Languages

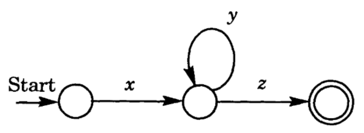
The Pumping Lemma for Regular Languages

- **The Pumping Lemma for Regular Languages**:

  Let L be an infinite regular language over the alphabet A. Then there exists an integer $m > 0$ (the number of states in a DFA recognizing L), such that for every string $s \in L$, with $|s| \geq m$, there exist strings $x, y, z \in A^*$, such that $s = xyz$, $y \neq \Lambda$, $|xy| \leq m$ and $xy^k z \in L$, for all $k \geq 0$.

  Since L is regular, it is recognized by a DFA. Suppose the DFA has m states. Consider a string s, with $|s| \geq m$. To accept s, the DFA must enter some state twice.

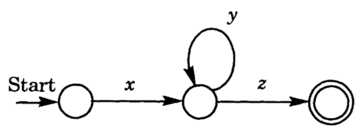
  So the DFA must follow a walk that contains a cycle as in the figure, where:



  - $s = xyz$;
  - each arrow represents a path that may contain other states of the DFA, with x, y, and z the strings of letters along each path;
  - y is the string on a single (the first) traversal of the cycle.

The Pumping Lemma for Regular Languages (Cont'd)

- We reasoned that the DFA accepting L, on input s follows a walk as shown on the right.



We have the following properties:

- Since $|s| = m$, the walk must traverse the cycle at least once. So $y \neq \Lambda$.
- Since the walks for x and y consist of distinct states (remember that y is the string on just one traversal of the cycle), it follows that $|xy| \leq m$.
- Since the walk through the cycle may be traversed any number of times, it follows that the DFA must accept all strings of the form xy^kz for all $k \geq 0$.

This property is called the **pumping property** because the string y can be pumped up to y^k by traveling through the same cycle k times.

Example

- Use the Pumping Lemma to show that $L = \{a^n b^n : n \geq 0\}$ is not a regular language.

  Assume, by way of contradiction, that L is regular.

  Consider the integer $m > 0$ of the Pumping Lemma.

  Take $s = a^m b^m \in L$, with $|s| = 2m > m$.

  By the Pumping Lemma, there exist strings x, y, z, such that

  $$s = a^m b^m = xyz,$$

  with $y \neq \Lambda$, $|xy| \leq m$ and $xy^k z \in L$, for all $k \geq 0$.

  - Since $|xy| \leq m$, x and y consist only of a's.
    So $y = a^n$, from some $n > 0$.
  - Since $xy^k z \in L$, for all $k \geq 0$, we have

    $$a^{m+n} b^m = xy^2 z \in L, \ n > 0.$$

    But this is a contradiction!

Example

- Show that the language P of palindromes over the alphabet $\{a, b\}$ is not regular.

  We assume that P is regular and try for a contradiction.

  Then there is a DFA with m states to recognize P.

  Choose a palindrome of the form $s = a^m b a^m$.

  Apply the Pumping Lemma to get strings x, y, z such that $y \neq \Lambda$, $|xy| \leq m$ and $xyz = s = a^m b a^m$.

  Since $|xy| \leq m$, x and y are both strings of a's.

  Thus $y = a^n$, for some $n > 0$.

  Since $xy^k z \in L$, for all $k \geq 0$,

  $$a^{m+n} b a^m = xy^2 z \in L, \ n > 0.$$

  This is a contradiction, since $a^{m+n} b a^m$ is not a palindrome!

Closure Properties of Regular Languages

- **Closure Properties of Regular Languages**:
  1. The union of two regular languages is regular;
  2. The language product of two regular languages is regular;
  3. The closure of a regular language is regular;
  4. The complement of a regular language is regular;
  5. The intersection of two regular languages is regular.

4. Let L be a regular language.

   Then L is the language accepted by a DFA D.

   Construct a new DFA, say D' from D by making all the final states nonfinal and by making all the nonfinal states final.

   If we let A be the alphabet for L, it follows that D' recognizes the complement $A^* - L$.

   Thus the complement of L is recognized by a DFA and is therefore regular.

Example

- Suppose L is the language over the alphabet $\{a, b\}$ consisting of all strings with an equal number of a's and b's.

  Show that L is not regular.

  Suppose to the contrary that L is regular.

  Let M be the language of the regular expression $a^* b^*$.

  Then $L \cap M = \{a^n b^n : n \geq 0\}$.

  We know that M is regular because it is the language of the regular expression $a^* b^*$.

  Therefore $L \cap M$ must be regular.

  In other words, $\{a^n b^n : n \geq 0\}$ must be regular.

  But we know that $\{a^n b^n : n \geq 0\}$ is NOT regular.

  Therefore, L is not regular.

Closure Under Morphisms of Regular Languages

- Let A be an alphabet and let $f : A^* \to A^*$ be a function.
- f is called a **language morphism** if the following conditions hold:
  1. $f(\Lambda) = \Lambda$;
  2. $f(uv) = f(u)f(v)$, for all strings u and v.
- Let $f : A^* \to A^*$ be a language morphism.
  Let L be a language over A.
  (a) If L is regular, then $f(L)$ is regular.
  (b) If L is regular, then $f^{-1}(L)$ is regular.

(a) Suppose L is regular.

Then it has a regular grammar.

We create a regular grammar for $f(L)$ as follows:

Transform productions like $S \to w$ and $S \to wT$ into new productions of the form $S \to f(w)$ and $S \to f(w)T$.

The new grammar is regular, and any string in $f(L)$ is derived by this new grammar.

Example

- Use Closure Under Language Morphisms to show that the language $L = \{a^n bc^n : n \in \mathbb{N}\}$ is not regular.

  We can define a morphism $f : \{a, b, c\}^* \rightarrow \{a, b, c\}^*$ by

  $$f(a) = a, \quad f(b) = \Lambda, \quad f(c) = b.$$

  Then $f(L) = \{a^n b^n : n \geq 0\}$.

  If L is regular, then we must also conclude by Closure Under Language Morphisms that $f(L)$ is regular.

  But we know that $f(L)$ is NOT regular.

  Therefore L is not regular.